Designing a Profiling and Visualization Tool for Scalable and In-Depth Analysis of High-Performance GPU Clusters

Pouya Kousha, Bharath Ramesh, Kaushik Kandadi Suresh, Ching-Hsiang Chu, Arpan Jain
Nick Sarkauskas, Hari Subramoni and Dhabaleswar K. Panda
Department of Computer Science and Engineering
The Ohio State University
{kousha.2, ramesh.113, kandadisuresh.1, chu.368, jain.575, sarkauskas.1}@osu.edu, {subramon, panda}@cse.ohio-state.edu

***Abstract*—The recent advent of advanced fabrics like NVIDIA NVLink is enabling the deployment of dense Graphics Processing Unit (GPU) systems, e.g., DGX-2 and Summit. The Message Passing Interface (MPI) has been the dominant programming model to design distributed applications on such clusters. The MPI Tools Interface (MPI_T) provides an opportunity for performance tools and external software to introspect and understand MPI runtime behavior at a deeper level to detect performance and scalability issues. However, the lack of low-overhead and scalable monitoring tools have thus far prevented a comprehensive study of efficiency and utilization of high-performance interconnects such as NVLinks on high-performance GPU-enabled clusters.**

In this paper, we address this deficiency by proposing and designing an in-depth, real-time analysis, profiling, and visualization tool for high-performance GPU-enabled clusters with NVLinks. The proposed tool builds on the top of the OSU InfiniBand Network Analysis and Monitoring Tool (INAM). It provides insights into the efficiency of different communication patterns by examining the utilization of underlying GPU interconnects. The contributions of the proposed tool are two-fold: 1) domain scientists and system administrators can understand how applications and runtime libraries interact with underlying high-performance interconnects, and 2)Proposed tool enables designers of high-performance communication libraries to gain low-level knowledge to optimize existing designs and develop new algorithms to optimally utilize cutting-edge interconnects on GPU clusters. To the best of our knowledge, this is the first such tool which is capable of presenting a unified and holistic view of MPI-level and fabric level information for emerging NVLink-enabled high-performance GPU clusters.

***Index Terms*—MPI, MPI_T, NVLink, GPU, Profiling**

I. Introduction and Motivation

Emerging high-performance computing (HPC) and cloud computing systems are widely adopting Graphics Processing Units (GPUs) to support the computational power required by modern scientific and machine learning applications. Offering high-bandwidth memory, tensor processing, and massive parallelism, GPUs enable running complex applications such as weather forecasting [1], brain data visualization [2], and molecular dynamics [3].

To further enhance and complement the high compute power of current hardware for applications, researchers are building large-scale GPU clusters with high-speed interconnect technology such as peripheral component interconnect express (PCIe), NVIDIA NVLink, AMD Infinity Fabric, Mellanox InfiniBand, and so on. The typical solution of scaling out and scaling up [4] has been proposed to meet the growing demand for more processing power by applications. Primarily, these systems are designed for applications that require dense computation. Recent studies show that such dense GPU systems like DGX-2 accelerate Deep Neural Network (DNN) training process [5]. Inevitably, such solutions will require more communication among GPUs through various interconnects. Therefore, high-speed interconnects that deal with complex intra-node and inter-node topology become vital components in the HPC ecosystem. In spite of the fact that computing components have evolved dramatically, the interconnect and network components do not keep up the same pace. To gain high-performance implementations, efficient data movement schemes between GPUs within a node, and across multiple GPU nodes is critical. Therefore, detailed insight into what is happening inside a node and between the nodes during communication is required to assist application users and developers in identifying and improving performance bottlenecks.

MPI is the de facto communication standard widely used in developing parallel scientific applications on HPC systems. Considering the recent advances in interconnect technology (NVLinks, X-Bus etc.), understanding the interaction between applications, MPI libraries, and the communication fabric becomes ever more challenging for system administrators, application and MPI designers. Although it is possible to observe the performance degradation of applications from an end-to-end perspective, finding out the root cause is not trivial.

As more features are getting introduced into MPI communication libraries, it is getting more complex to track the performance of various components inside the MPI library as well as to select the best communication pattern to use based on the message size and cluster specifications. Extracting maximal performance from a communication library such as MPI requires tuning it extensively to choose a specific algorithm for a given situation. Typically, such tuning is done by system administrators or the MPI library provider using microbenchmark suites [6], which need not be representative of communication patterns in real applications. In this context, the "tuned" algorithms selected based on microbenchmark

2640-0316/19/$31.00 ©2019 IEEE
DOI 10.1109/HiPC.2019.00022

level evaluations may not provide the best performance at the application level. Moreover, it is time-consuming and tedious to conduct tuning for all different applications on various systems. Broadly, this leads to the following challenges for designers of high-performance applications and communication middleware: 1) how should they identify what the root causes are for performance degradation? 2) how should they know if the performance issue (if any) is due to problematic designs in the communication library or the application or even a hardware issue? Moreover, for an MPI designer, understanding the internal interaction and inter-dependencies between different components of the MPI library and the correlation of MPI level and with network/hardware level can play a key role in easily identifying bottlenecks, which leads to significantly optimized designs.

Currently, users and administrators of HPC systems and designers of HPC applications/middle-ware rely on a plethora of tools to aid them in this interplay. There exists a variety of MPI level profiling tools (TAU [7], HPCToolkit [8], Intel VTune [9], IPM [10], mpiP [11]) that give insights into the MPI communication behavior of applications. However, they are unable to profile emerging communication fabrics like NVLink. On the other hand, several network level profiling and analysis tools exist that allow system administrators to analyze and inspect the network fabric (Nagios [12], Ganglia [13], Mellanox Fabric IT [14], BoxFish [15], Lightweight Distributed Metric Service (LDMS) [16]). However, they are unable to relate activity in the communication fabric to their triggering events in the MPI library. Thus, a fundamental gap exists in the current class of monitoring tools, making the correlation of MPI events to network activity cumbersome at best.

The MPI forum [17] has been actively working on bridging this gap with the MPI_T [18] interface. The MPI_T interface allows tools to interact with the MPI library through control and performance variables. Researchers have already begun to take advantage of this interface to provide optimization and tuning hints to the users [19]. However, these tools have no knowledge about the underlying network fabrics and thus suffer from the same drawbacks as other existing MPI tools.

These issues in the capabilities of existing profiling tools lead us to the following broad challenge: **How can we design a tool that enables in-depth understanding of the communication traffic on the interconnect and GPU through tight integration with the MPI runtime?**

A. *Contributions*

This paper presents a real-time and scalable profiling tool, built on top of CUDA Profiling Tools Interface (CUPTI) [20] and OSU INAM [21], that bridges the gap between MPI software counters and GPU-related hardware counters. Figure 1 depicts the high-level overview of our design. The green boxes represent layers of the stack in which the tool is implemented. The gathered metrics from various levels are inserted into the storage component. Inside the web-based UI, the metrics from each level are correlated to each other and are shown. In this paper, we make the following key contributions:

- Design a profile-enabled communication library for GPU communication to gather GPU and MPI performance counters through CUPTI and MPI_T interfaces, respectively
- Present a real-time and low-overhead profiler tool to correlate MPI-level with network-level metrics
- Introduce MPI_T event-based metrics for MPI library for point-to-point and collective communication patterns
- Add performance variables (PVARS) to accurately quantify MPI communication operations
- Evaluate the performance and efficacy of the proposed tool on a modern multi-GPU system with NVLink interconnect
- Demonstrate the capability to analyze and classify the traffic and link utilization for MPI communication patterns for each GPU interconnect like NVLink

To the best of our knowledge, this is the first such tool which is capable of presenting a unified and holistic view of MPI-level and fabric-level information for emerging NVLink-enabled high-performance GPU clusters.

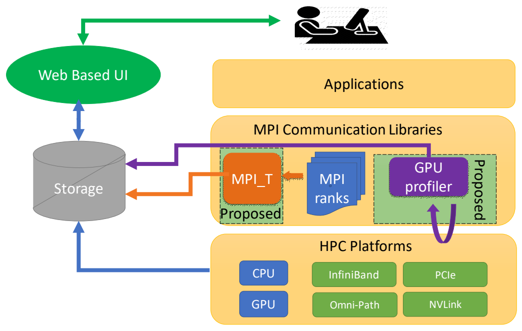


Fig. 1. High level overview of the proposed tool - Green boxes

II. Background

A. *Interconnect Technology*

NVLink and/or PCIe interconnects are used to connect CPU/GPU to GPU and InfiniBand or Ethernet to connect nodes in modern GPU systems.

PCIe is a high-speed standard bus interconnect for high-speed components like CPU, GPU and IB Host Channel Adapters (HCAs). With the introduction of each generation of PCIe, the bandwidth got doubled as generation-1 starting at 2GB/s and generation-4 featuring 16 GB/s unidirectional bandwidth. As the community is moving toward scaling up, NVIDIA introduced NVLink interconnect, primarily focusing on improving the connectivity of GPUs and CPU to GPU. DGX-2 systems offer NVLink2 connectivity which gives 25GB/s unidirectional bandwidth and multi-link NVLink2 inter-GPU connections inside a single node. InfiniBand is a high bandwidth, and low latency interconnect for HPC systems. The latest InfiniBand Enhanced Data Rate (EDR) and upcoming High Data Rate (HDR) adapters offer bidirectional bandwidth of 100Gbps and 200Gbps, respectively. As new interconnect technology are introduced, the next generation HPC systems, including the #1 on Top500 (top500.org) *Summit* are deploying NVLink-enabled dense GPU nodes with InfiniBand interconnects.

B. *Low-level Profiling Tools Interface for GPU*

The CUDA Profiling Tools Interface (CUPTI) developed by NVIDIA provides the API for tracing and profiling that target CUDA applications. CUPTI provides the following five APIs categories: the activity API, the callback API, the event API, the metric API, and the Profiler API. Activity APIs enable asynchronous gathering of GPU or application's CPU activity. They are used to discover NVLink and PCIe topology. The callback APIs enable registering callbacks in the code. The event APIs enable to query, configure, start, stop, and read the event counters on a CUDA device. The metric APIs collect information calculated from the events. The Profiler APIs help to find out the average, maximum and minimum of some metrics. As of now, the Profiler API does not support NVLink metrics. Using these APIs, one can develop profiling tools that give insight into the CPU and GPU behavior of CUDA applications.

C. *MPI Tools Information Interface (MPI_T)*

The MPI Tools Information Interface (MPI_T) provides a standard mechanism for MPI tool developers to both inspect and tweak the various internal settings and performance characteristics of MPI libraries. The MPI_T interfaces define two types of objects. The first type of object is the performance variable (PVAR). Accessing the values of performance variables allows the software to peak under the hood of the MPI library to determine the state and how it is being affected by the MPI application. The second type of object is the control variable (CVAR). This type of object is tied to a modifiable parameter of the MPI library. Accessing and modifying these variables will allow the software to change the behavior of the MPI library.

III. Designing a High-Performance, Low Overhead, and Scalable GPU Profiling Library

In this section, we elaborate on and discuss the proposed design of our tool. It has three major components a) designing a library to collect NVLink metrics and gather MPI_T information, b) the storage component, and c) the visualization interface. Figure 2 depicts the detailed design and the interplay between the different modules of the proposed tool. Note that the same designs used in this paper to discover and gather metrics for NVLink are applicable for PCIe interconnects as well [1].

[1]There are issues with the current release of CUPTI accessing PCIe metrics. It has been confirmed by NVIDIA developers at https://devtalk.nvidia.com/default/topic/1051538/cuda-profiler-tools-interface-cupti-/empty-uuid-for-pcie-records/.

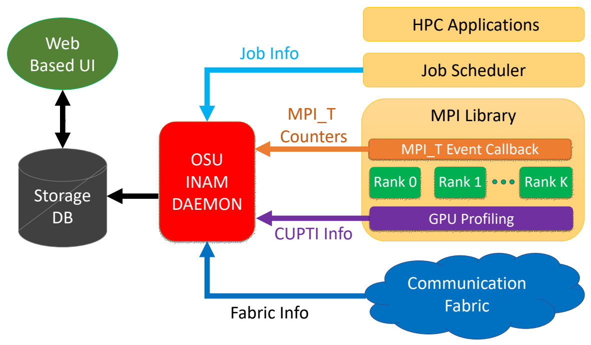


Fig. 2. Detailed overview of proposed design

A. *Profiler Interface for MPI+CUDA Communication*

We propose two primary profiler modules on top of the CUPTI and MPI_T interfaces to enable profiling the GPU and MPI activities over high-performance interconnects. We implement the profiling interfaces in a CUDA-Aware MPI library, MVAPICH2 [22]. The design can be applied to any MPI library.

1) Low-overhead GPU Profiler Module: The proposed GPU profiler has two primary components, a) intra-node topology discovery and b) metrics inquiry.

GPU Topology Discovery: The topology of GPU configuration is vital for profiler tools to report and correlate the hardware metrics with corresponding MPI processes. To discover intra-node topology with lower overhead, we query the GPU and NVLink connectivity by using appropriate CUPTI activity APIs and callback arguments (e.g., `CUPTI_ACTIVITY_KIND_NVLINK`) during the initialization phase of the communication runtime (e.g., MPI_Init in the context of MPI). NVLink Activity APIs in CUPTI reports the connectivity in terms of logical NVLinks. Each logical NVLink can contain up to six physical NVLinks on the latest NVIDIA GPU architecture [23]. First, we query information such as endpoints, device types, ports, bandwidth, and universally unique identifiers (UUID). Next, we use the UUID of endpoints of each physical NVLink and iterate through the device activities that have been discovered to identify the corresponding device name and other required information and save this information into a data structure which will help us to correlate metrics and physical NVLinks in the profiler module. To minimize the query overhead, only one process in each node queries and caches the topology information. For a node with 4 GPUs, the topology discovery takes 0.25 second on average. The discovered intra-node topology are stored in a shared memory region that is accessible by all other MPI processes within the node.

GPU Metrics Inquiry: Once the intra-node GPU topology has been discovered, the proposed module continually queries the GPU metrics through CUPTI APIs. We provide the flexibility to set the query interval through configuration files to the user. To ensure that the profiling done by our proposed tool does not affect the performance of the application and to minimize the resource consumption of the profiler tool, we propose to offload the task to a separate *profiler thread*. The profiler thread will be responsible for querying metrics for all the GPUs within the node. The goal is to offload reading the metrics to the profiler thread on the host to keep the overhead as low as possible on CPU and GPUs. The abundance of CPU cores on modern platforms and the fact that most (if not all) applications only use one process per GPU, will ensure that the profiler thread will have enough compute resources without affecting application performance. Figure 3 depicts the structure and flow of this design.

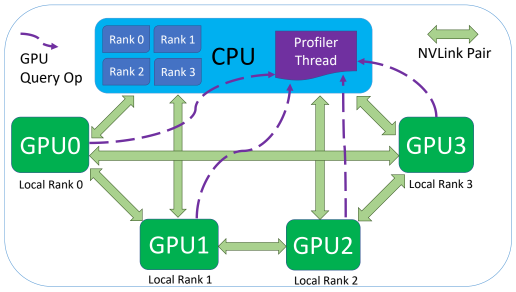


Fig. 3. Collection of intra-node GPU metrics

Our proposed design has three phases to gather metrics: startup, query, and exit.

- *Startup Phase*: Once the MPI runtime detects of the use of GPU, a shared region of system memory across MPI ranks is used to hold the data structures for exchanging data between the profiler thread and MPI ranks. Each MPI rank updates the shared memory region with the assigned device ID, local and global MPI rank information. Then, one MPI rank on each node acts as the leader rank (e.g., rank 0), launches the profiler thread and updates the number of devices that are participating in the communication. CUPTI requires a unique CUDA context to be used for the thread that activates the CUPTI event groups and reads the CUPTI events, and the CUDA context cannot be shared between the MPI ranks. Thus, the profiler thread creates and saves a CUDA context for each GPU to be used by the profiler thread. Note that this is a blocking collective operation within a node to prevent inconsistent and incorrect correlations later.
- *Query Phase*: The profiler thread loads the context of each participating GPU associated with an MPI rank and reads the CUPTI metrics in the background for all physical NVLink instances. For each GPU, the profiler thread will gather and map the metrics of each instance to physical NVLink instances by using the properties of the corresponding GPU that are discovered at the initialization phase. Next, the profiler thread sends the queried data along with a timestamp to the storage component. Once the queried metric data being stored, the profiler thread checks if the leader process (described above) has requested to stop profiling. If not, the profiler thread is put to sleep mode for the interval that the user has defined. Figure 2 shows the overall design and the interaction between components.
- *Exit Phase*: when the MPI ranks finish their tasks, i.e., the application calls MPI_Finalize in the MPI context, the local leader rank signals the profiler thread to stop profiling, and the thread will query and output the metrics for the last time and exit. This is to ensure that in case that one rank will finish faster, we capture the metrics for other ranks. Finally, MPI ranks disable and destroy the CUPTI event groups.

Since CUPTI gives us the metrics for all NVLinks, we depend on the information provided by the MPI processes to get the necessary information for the GPUs that are being used. The remaining information can be reported by the user's choice or discarded. Although we only used NVLink metrics for our paper, such structure can be used to get any CUPTI events or metrics. The user needs to pass the list of metrics to be monitored as a run-time parameter.

2) Extending MPI_T Performance Variables: The proposed GPU profiler described in Section III-A1 gathers various metrics that will allow users to understand the utilization of interconnects like NVLink. In this section, we discuss the necessary MPI counters and their implementation in MPI runtime using MPI_T interface to enable correlation between GPU metrics and MPI communication.

In this work, we focus on tracking performance characteristics for MPI point-to-point and collective communications inside the MPI library using MPI_T performance variables (PVARs). For each collective and point-to-point operation, every rank stores the total bytes sent to and received from every other rank, an array of start and end time-stamps, selected algorithm for the communication, and the number of times a particular algorithm/function was called. If there are K MPI ranks, each rank tracks performance variables using a $N * K$ matrix, where N is the pre-defined value to keep the maximum number of PVARs to be stored before it is flushed. In the case of time-stamps, we record a fixed set of start and end times for each of the K ranks in the matrix. Time-stamps are measured in micro-seconds. The time-stamps wrap around the call to the communication operations inside the MPI library. This information can be used to find the time periods when an MPI operation utilizes the network.

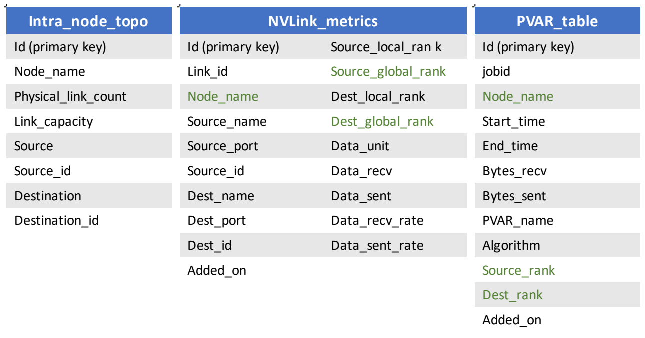
To gain insights into the interconnect utilization of different communication schemes in the MPI runtime, we use MVAPICH2 as an example and add new PVARs for different point-to-point primitives and collective algorithms. To minimize the overhead in gathering data, PVAR information is only sent if the aggregated bytes sent for a particular MPI operation exceeds a user-specified threshold. The user can define a granularity (ranging from byte to gigabyte) to filter data collection. This is useful for the users that are interested in significant changes in the communication. By allowing the user to define such a threshold, we reduce the amount of data that is being collected and stored. Therefore, our tool can provide more historical data as well as reduce the time to query the database and get the results, as discussed in the result section.

B. Storage and Correlation of MPI_T and GPU Metrics

In this section, we describe storage interface and how we correlate the metrics from MPI_T and the GPU.

1) Database Schema and Storage Interface: Figure 4 depicts the database schema used to store the GPU metrics and PVAR information into MySQL. The *pvar_name* and *pvar_algorithm* columns are used to represent a particular MPI_T PVAR variable associated with MPI operations like allreduce.

The GPU related information are stored into two different tables, *intra_node_topology* and *nvlink_metrics*. The *nvlink_metrics* table stores the metrics for each physical NVLink. For GPU metrics, we distinguish between NVLinks using *source_id, source_port, destination_id* and *destination_port* where destination/source_id refer to the device ID. The *intranode_topology* table stores the link-level information such as source, destination, the total number of physical links between the source and destination and link capacity. Such a database scheme design allows the visualization module to quickly render the intra-node topology while fetching the GPU metrics.



Intra_node_topo	NVLink_metrics		PVAR_table
Id (primary key)	Id (primary key)	Source_local_ran k	Id (primary key)
Node_name	Link_id	Source_global_rank	jobid
Physical_link_count	Node_name	Dest_local_rank	Node_name
Link_capacity	Source_name	Dest_global_rank	Start_time
Source	Source_port	Data_unit	End_time
Source_id	Source_id	Data_recv	Bytes_recv
Destination	Dest_name	Data_sent	Bytes_sent
Destination_id	Dest_port	Data_recv_rate	PVAR_name
	Dest_id	Data_sent_rate	Algorithm
	Added_on		Source_rank
			Dest_rank
			Added_on

Fig. 4. Database schema used for storing the metrics gathered by the tools

2) Correlating MPI_T and GPU Metrics: The correlation of MPI and GPU metrics happens in the web based UI of OSU INAM. After storing the data, we use the fields colored green are used in Figure 4 to read and map the NVLink metrics and PVAR metrics for a specific node. A combination of Node_name, source_rank, and destination_rank correlates the PVARs from MPI level to the NVLink metrics from network level. Please note that the PVAR information are mapped to a logical NVLink since we cannot distinguish between physical NVLinks at the MPI level. Then, the charts in Section IV will use an user-defined intervals to filter the data based on added_on timestamps. At this level of development of the tool, the user can correlate them using timeline in X-axis.

IV. New Visualization Components and Features Introduced

In this section, we list the features and visualization components of OSU INAM to incorporate the changes brought in by our proposed design described in Section III

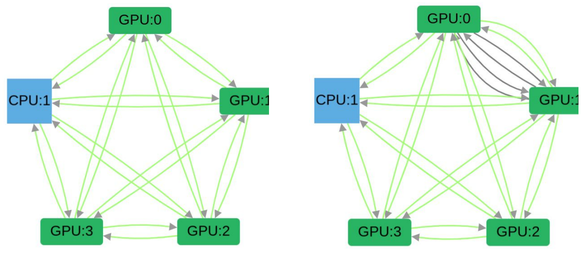
The proposed tool is able to present information in a unified and holistic fashion to the user. This allows end users like application scientists and developers of high-performance communication libraries to identify the impact the algorithms have on the communication links and the extent to which they are able to saturate the link bandwidth. In the absence of such a tool, the end users would have had to rely on multiple tools as indicated in Section I and do a lot of manual calculations to arrive at the same conclusion. Since the tool is tightly integrated with the MPI library with a focus on low overheard, it is less demanding on the end application when compared to external tools. Thus, the proposed solutions are able to deliver benefits greater than just sum of both MPI_T and CUPTI interfaces.

A. New Tool Capabilities Introduced

The OSU INAM web application provides the capability to see job level information including historical jobs and live jobs. For the jobs, we have the number of calls to a MPI operation for different message sizes, the number of occurrence, and the latency of each call as PVAR table in Figure 4. Bytes_sent and Bytes_recv show the amount of data transferred between two MPI ranks at the given timestamps. The latency can be calculated using start_time and end_time. Using the tool, the user can observe the network level link utilization and the amount of data transmitted for a MPI operation.

There are two ways to navigate through the intra-node page for a specific node. 1) search for the node in network view that shows the live inter-node view of the network or 2) choose the job from live jobs page where it shows the list of nodes involved.

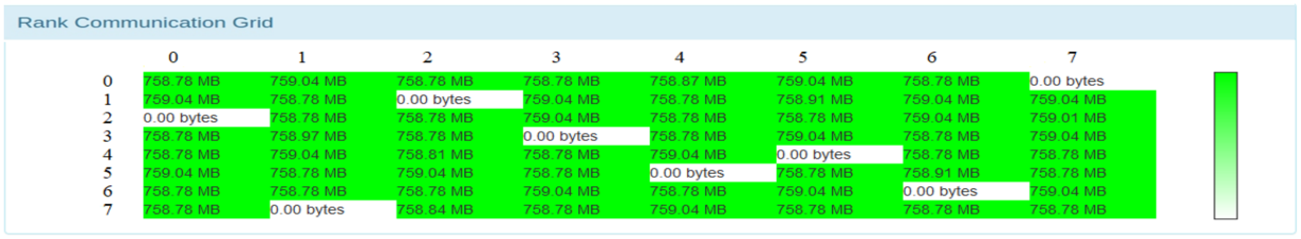
The intra-node page has the following features that are listed as items on the front-end interface :



(a) Logical view (b) Physical view of one NVLink

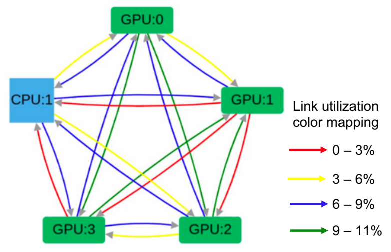
Fig. 5. Intra-node topology visualization for one node

- **Node Topology** : This feature depicts the Intra-node topology information on the front-end. Figures 5(a) and 5(b) show the connectivity of GPUs and CPUs in two forms: logical and physical NVLinks. User can expand the logical NVLinks to view the link utilization of each physical link. Figure 5(b) shows the physical NVLinks between GPU0 and GPU1. Each link changes color based on its link utilization level. The coloring for topology sketches are as follows: Grey represents 0%, light green up to 25%, dark green from 25% to 50%, orange from 50% to 75%, and red from 75% to 100% of the total available bandwidth. This information is

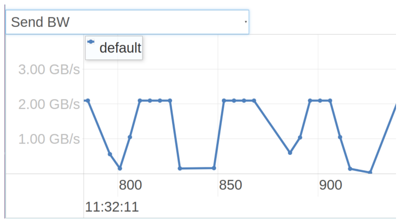

Rank Communication Grid

	0	1	2	3	4	5	6	7
0	758.78 MB	759.04 MB	758.78 MB	758.78 MB	758.87 MB	759.04 MB	758.78 MB	0.00 bytes
1	759.04 MB	758.78 MB	0.00 bytes	759.04 MB	758.78 MB	758.91 MB	759.04 MB	759.04 MB
2	0.00 bytes	758.78 MB	758.78 MB	759.04 MB	758.78 MB	758.78 MB	759.04 MB	759.01 MB
3	758.78 MB	758.97 MB	758.78 MB	0.00 bytes	758.78 MB	759.04 MB	758.78 MB	759.04 MB
4	758.78 MB	759.04 MB	758.81 MB	758.78 MB	759.04 MB	0.00 bytes	758.78 MB	758.78 MB
5	759.04 MB	758.78 MB	759.04 MB	758.78 MB	0.00 bytes	758.78 MB	758.91 MB	758.78 MB
6	758.78 MB	758.78 MB	758.78 MB	759.04 MB	758.78 MB	759.04 MB	0.00 bytes	759.04 MB
7	758.78 MB	0.00 bytes	758.84 MB	758.78 MB	759.04 MB	758.78 MB	758.78 MB	758.78 MB

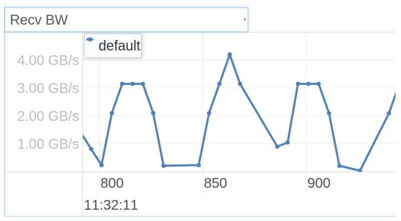
Fig. 6. Rank level communication grid, each element in the grid represents amount data transferred from rank i to rank j where i,j is the element's position in the grid. This figure is showing a Allreduce operation happening between MPI ranks



(a) Intra-Node link usage with NCCL



(b) NVLink utilization GPU0 to GPU1



(c) NVLink utilization GPU1 to GPU0

Fig. 7. Graphs for NCCL Allreduce test. For the charts, the X-axis represents time and Y-axis represents the bandwidth utilization for the NVLink

obtained from intra-node_topology table. Since NVLink does not provide CPU level information, currently all CPUs are shown as a single block. The percentages and colors are user configurable.

- **NVLink charts** : This feature shows the metrics obtained from CUPTI. The metrics include data_sent, data_received, send_rate, recv_rate. Figure 7(b) shows the plot of the CUPTI metric: 'Send BW'obtained on the link from GPU0 to GPU1. The plot can be live or historical. The metric can be changed using a drop-down menu. The data is obtained from the nvlink_metrics table.
- **PVAR counters chart** : This chart provides the plot of the PVAR countesr as shown in Figure 9(a) for each rank. The legend in the chart $Allreduce - rd$ shows the type of PVAR.
- **Rank level communication grid** : This shows the data transferred between each pair of rank in the current node, as shown in Figure 6.

V. Usage Scenarios

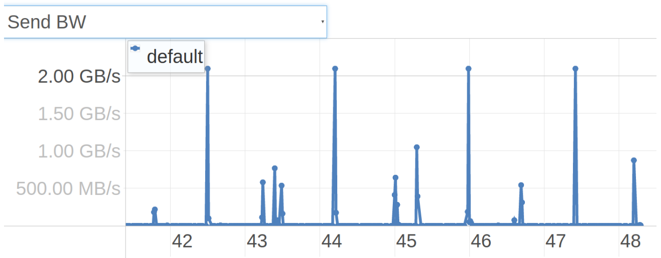
In this section, we aim to highlight how different classes of users can take advantage of our tool to gain in-depth insights into the communication performance of applications using different communication middleware. In particular, we focus on two popular high-performance GPU-aware communication middleware a) the NCCL2 library from NVIDIA and b) the TensorFlow Deep Learning framework running over MPI.

A. Understanding the Communication in NCCL Ring-based Allreduce

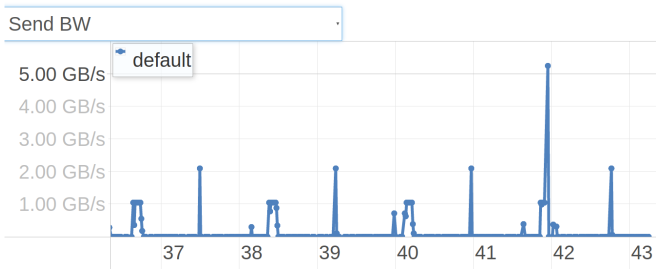
We ran the NCCL library with MPI library for Ring-based Allreduce. Figure 7(a) shows the topology on one of the nodes with links being colored based on their link utilization. As expected the link utilization is not to the full capacity since there is computation happening with communication. In Figure 7(a), we see that the links in the clockwise direction (From CPU:1 to GPU:3) are either colored yellow or red indicating that the usage is relatively low whereas the links in the counter clockwise direction are either blue or green indicating that the usage of these links is relatively high. From this, we can infer that the links in one direction have a higher utilization than links in the other direction showing an ineffective use of bi-directional bandwidth in the given ring communication pattern. This gives developers valuable feedback into how the algorithm has been designed. Further, by comparing Figures 7(b) and 7(c) provided by our tool and OSU INAM GUI for NVLink utilization, we can clearly see a difference in how the NVLinks are being utilized. For example, the link from GPU1 to GPU0 has, on average, higher bandwidth compared to the link from GPU0 to GPU1. The same observation for GPU1 to GPU2 and so on. Thus, it can be concluded that for the case where communication's direction is clockwise as in the case when GPU 3 wants to send data to GPU 2, the data is probably moved counterclockwise from more chain hops (CPU and GPU). This is because the algorithm only utilized one direction of the ring while it could have utilized the opposite direction. This case shows that even without having the MPI_T-based PVAR information available, the user can use our tool to understand the trends and interactions between network level components for other applications that are not using MVAPICH as communication library.

B. Understanding the Communication of MPI-based TensorFlow

We ran Tensorflow version 1.12 with MVAPICH2 to gather both PVARs and GPU metrics for distributed training using Horovod. Horovod is a distributed DNN training framework

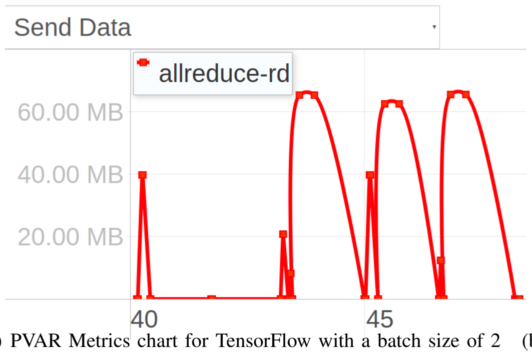


(a) NVLink Metrics chart for TensorFlow with a batch size of 2

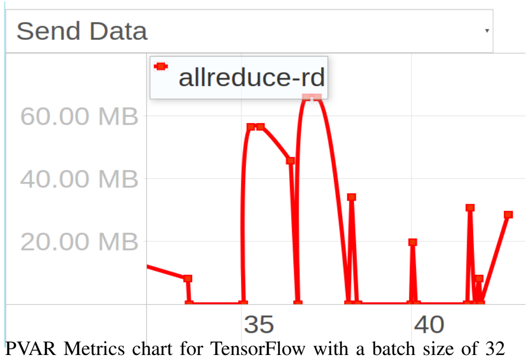


(b) NVLink Metrics chart for TensorFlow with a batch size of 32

Fig. 8. NVlink charts for GPU0 to CPU for TensorFlow test with Resnet50 model with different batch sizes. The X-axis represents time and Y-axis represents the bandwidth utilization for a link



(a) PVAR Metrics chart for TensorFlow with a batch size of 2



(b) PVAR Metrics chart for TensorFlow with a batch size of 32

Fig. 9. PVAR charts for rank0 to rank2 for TensorFlow test with Resnet50 model with different batch sizes. The X-axis represents time and Y-axis represents the size of the message sent over the network

for TensorFlow, Keras, PyTorch, and MXNet. The experiment is executed across two nodes with four GPUs per node with different batch sizes for the same number of iterations(batches).

Figures 8(a) and 8(b) show the experiment with Resnet50 model for NVLink metrics. We ran the experiments using batch sizes 2 and 32 for 5 batches. The users are usually interested in number of images per seconds. The goal of the test is to measure the performance of communication for the same model using different batch sizes. The smaller batch size will result in lower link utilization and therefore is less communication efficient.

Figures 9(a) and 9(b) show the MPI level PVAR information for the GPU0(rank0) to GPU2(rank2). From the figures we note that the peak data (message size) transferred between ranks are the same, but showing different patterns between ranks. From both figures, we can infer that Horovod uses certain message sizes in the allreduce operations and it depends on the Deep Learning model, batch size, GPU architecture, and other Deep learning parameters.

VI. Performance Evaluation

In this section, we present the evaluation of the overhead of GPU profiling and MPI_T and show how it scales. All the timings in this section are measured using monotonic clocks.

A. Experimental Setup

In this paper, we conducted the evaluation on an NVLink-enabled GPU system, where each node has two IBM POWER9 CPUs on 2 sockets connected via X-Bus, and each socket is connected to two NVIDIA Tesla V100 GPUs using NVLink2. Every socket has 22 CPU cores with four hardware threads per core. Red Hat Enterprise Linux Server release 7.5 with a kernel version of 4.14.0-49.18.1 is installed. Mellanox InfiniBand dual port EDR socket-direct adapters are connected through two 8-lane PCIe Gen4 interfaces. Mellanox OFED 4.3 was used on all nodes.

B. Overhead Analysis

The overheads of profiling for GPU profiler thread and MPI_T component are evaluated step by step. We describe the overheads from intra-node level to across nodes. In the end, we discuss the overall overhead and scalability of our design running Allreduce benchmark. We conducted the experiments on 8 nodes with 4 GPUs per node running an Allreduce micro-benchmark [6] between 8KB to 256MB message size range unless otherwise stated and used MVAPICH2 as the communication library. All the timings are gathered using high resolution(nsec) monotonic clock.

1) Quantifying the Overhead of GPU Profiling: In this section, we run our tool to see the detailed overhead inside/of each step of the GPU profiler. Table I describes the timing of each component at microsecond granularity for the startup phase. These steps need to be done for each GPU on the node once. On average, the startup overhead for each GPU in a node is around 4 millisecond(ms) excluding context creation. Table II shows the timing of CUPTI profiler thread per node for each phase. As we can see, the CUDA context creation in the profiler start-up makes the start-up expensive. The median of profiling query for each node with 4 GPUs is 1.299 millisecond. Figure 10 shows the dispersion of the querying data for profiler thread for 15,000 samples. As can be observed, the query timing does not have much variation.

TABLE I
TIMING IN STARTUP PHASE FOR EACH GPU IN MICROSECOND EXCLUDING CONTEXT CREATION

Metrics	Average	Min	Max	STDEV.p
context switch	9.81	4.16	32.41	8.14
metric properties and info	1464.56	147.68	31641.51	5454.41
event group creation	287.05	45.91	1349.08	417.01
event activation	2480.56	395.53	30454.41	6582.02
collection mode	1.38	0.66	4.23	1.05

TABLE II
TIMING OF THE GPU PROFILER THREAD PHASES FOR EACH NODE. EACH NODE HAS FOUR GPUS

Metrics	Average	Min	Max	STDEV.p
Startup phase	1.632 s	1.561 s	1.672 s	0.035 s
CUDA context creation	1.624 s	1.548 s	1.663 s	0.035 s
Query phase	1.37 ms	1.25 ms	35.18 ms	0.67 ms
Exit phase	88 us	85 us	93 us	28 us

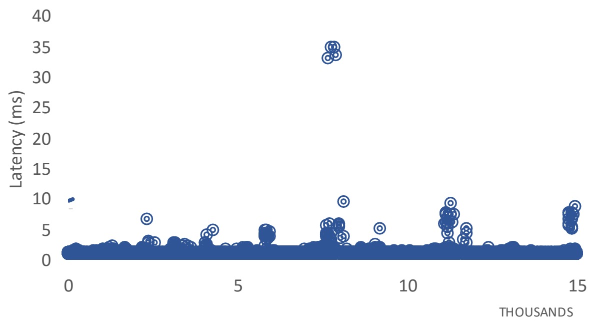


Fig. 10. Histogram of querying samples - each sample is the query time taken for all GPUs metrics in node using monotonic clock. There are some abnormalities but the overall timing is the around 2ms

2) *Measuring the Scalability of the GPU Profiler:* As our second test we varied the number of participating GPUs from one to four per node in Allreduce test to understand the impact of the number of GPUs in a node on the performance of the GPU profiler of our tool. Figure 11 shows that as the GPU count per node increases, the overall profiling time for all the GPUs in the node is increasing linearly (Number of GPUs per Node * time to profile one GPU) + a constant related to context switching on the profiler thread between GPUs on the node.

3) *Overhead of PVAR Collection:* We measure the overhead of collecting introduced PVAR metrics for each MPI communication operation. The PVARs collections are local for each MPI rank and collected in parallel for each MPI rank independently. Table III shows the stats for PVAR data collection measured on MPI rank 0 for 9,800 samples for an MPI operation. Our experiments showed that the timing overhead is approximately the same for the other ranks and are independent of the MPI operation. Therefore, we do not differentiate between them in the experiment. The variation in the timing can be explained by the use of a system call function to collect UTC timestamps, which is relatively expensive. The system call overhead can be reduced by using CPU cycle counters as done in libraries such as perftest from linux-rdma. We will implement this in the future.

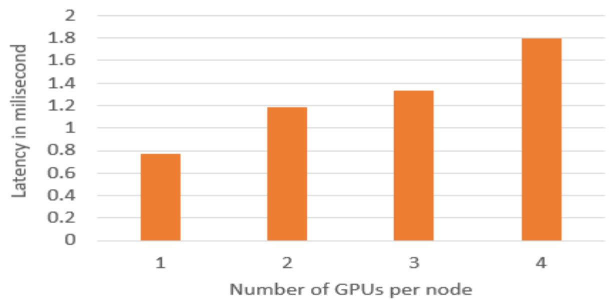


Fig. 11. Time to query varying number of GPUs inside a node

TABLE III
OVERHEAD OF COLLECTING PVAR DATA AT NANOSECOND GRANULARITY

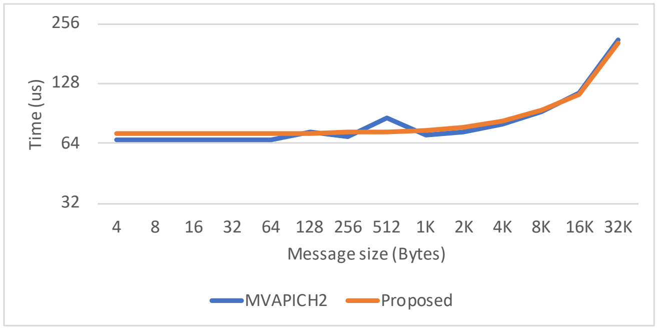
Metrics	Average	Min	Max	STDDEV.p
Collecting PVARs	517.63 ns	140 ns	16,204 ns	305.91 ns

4) *End-To-End Profiling Overhead:* To measure the overall overhead caused by profiling at the application, we run the osu_allreduce device to device CUDA collective test from the OSU micro-benchmarks software suite. We chose osu_allreduce benchmark since the benchmark is communication and computation intensive. Allreduce is the main and mostly communication operation used on Deep Learning applications. We compare it against the default version of MVAPICH2 and our proposed design. The test was run on 4 GPU nodes with 4 processes per node for at least 100 iterations. From Figure 12(a), we observe a 5-10% degradation for message sizes between 4 - 4,096 bytes. However, as seen in Figure 12(b), the gap closes for large message sizes due to the fact that the overhead of profiling is dominated by the variation in collective performance.

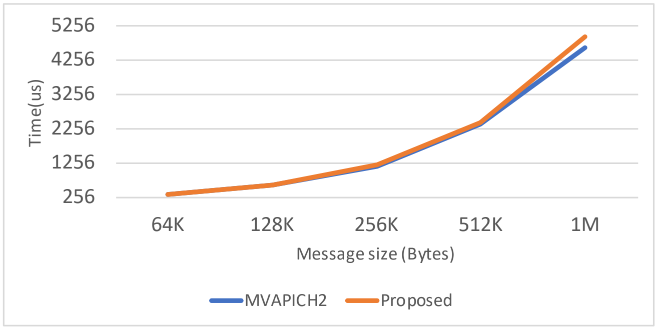
C. *Impact on Database Performance*

Figure 13 displays the trade-off between the user-defined data insertion granularity and the insert time to the database. The data was inserted with a bulk insertion into MySQL using mysql_query().

As the granularity decreases, less rows are to be inserted which decreases the time required. In the figure, there is negligible difference between a 1 byte granularity and a 1 KB granularity due to the data sample not having any points that are below 1 KB. A granularity setting of 10 MB inserts about two and a half times faster than a granularity setting of 1 KB. Note that there are no MySQL reads in this benchmark.



(a) Small and eedium message range



(b) Large message range

Fig. 12. Comparison between MVAPICH2 and proposed design on OMB allreduce - The X-axis represents message size and Y-axis represents the time taken in micro-seconds

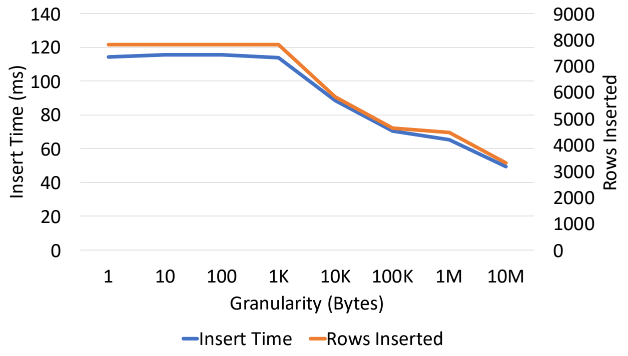


Fig. 13. Impact of data insertion granularity and database insert time

VII. Proposed Enhancements for MPI_T Standard

In this section, we discuss some issues we faced in designing an MPI library agnostic profiling tool that relies on the MPI_T interface and some suggestions to the MPI Forum on how these can be addressed in the standard to enable easier development of similar tools in the future.

Ideally, the goal is to design a profiling interface that is agnostic to MPI libraries and the application stacks so that the users can do a pre-load and use the tools with the minimum overhead. Currently, MPI_T events interface is under discussion and is likely to be included in the next MPI standard. While the current standard draft explains the functionality that should exist for the event interface, there are no standards for the implementer on naming the PVARs. As a result, different profiling tools and libraries will come up with their own naming and description for events and PVARs. Therefore, profiling tools that develop for library "X" may not be portable to library "Y" even though that the semantics of profiling metrics is the same.

We propose the need for MPI community to come up with a standard PVAR and event description so that, different tools implementations would be portable to each other. For example, most MPI libraries have different algorithms/implementations for a given collective communication operation, say, broadcast. The total number of algorithm available for broadcast and the selected algorithm can be reported through some PVARs (e.g. "MPI_T_BCAST_NUM_ALG" and "MPI_T_BCAST_ALG" respectively). In this way, a user knows what metrics they require to profile and can get the ID and specifications required for it from the profiler using the standard functions. Such standardization would allow other vendors to develop their profiling tools accordingly and enhance portability.

VIII. Related work

In the world of supercomputing, as it stands today, there is a need and lack of full-stack monitoring tools. Monitoring MPI jobs down to the port counters are essential for debugging job failures and finding network bottlenecks. In this section, we give details of existing tools (in literature and those available as products) other than those that we have already mentioned in Section I. The Texas Advanced Computing Center has developed a tool called *TACC STATS* [24]. This tool correlates job and system level statistics to create reports. The reports are used to identify issues in the jobs or systems infrastructure, but, not in real time. In [25], the authors proposed a suite built on MPI and allows for different metrics for each job and thus may not provide a full system view of how the jobs are interacting with the fabric. The authors in [26] describe a library that goes beyond the PMPI interface to gather MPI states and lower-level network statistics. Paraver [27] is used to provide visualization of the data. However, the tool could not visualize / model network activities. NVIDIA released SASSI [28] to support fine grained analysis tool for GPUs but it is closed source and has issues like portability, complexity and the instrumentation level is low for the developers. Authors of [29] proposed a profiler on the top of LLVM to instrument CUDA code, however, it does not provide insight into MPI level. As described in Section I, there are several tools that allow systems administrators to analyze and inspect high-performance networks such as Mellanox, FabricIT, BoxFish, Nagios, or Ganglia. With all of these tools, there is no integration with the MPI Library, so correlating network traffic to MPI jobs is a manual endeavor. On the other hand, there are several tools such as TAU, HPCToolkit, Intel VTune, IPM, and mpiP that focus on the MPI applications but do not show the network traffic of the MPI job.

IX. Software Availability and Deployment

OSU INAM Tool v0.9.4 is available for free download and use from the project website [30]. So far this version had over 550 downloads from the project site. OSU INAM is currently deployed at the Ohio Supercomputer Center to monitor multiple HPC clusters. We are working with other institutions to deploy OSU INAM on their clusters. The designs implemented in this paper will be available in a future release of INAM.

X. Conclusion

In this paper, we proposed a real-time scalable analysis, profiling, and visualization tool for GPU-enabled clusters with NVLinks. Built on top of INAM, CUPTI and MPI_T interfaces, our tool provides insights into the efficiency of different communication patterns by examining the utilization of underlying GPU interconnects while having negligible overhead. It can also correlate correlate MPI-level with network-level metrics by taking advantage of MPI_T event-based metrics obtained from the MPI library for point-to-point and collective communication patterns and the fabric information gathered through CUPTI. To the best of our knowledge, this is the first such tool which is capable of presenting an unified and holistic view of MPI-level and fabric level information for emerging NVLink-enabled high-performance GPU clusters. The proposed solutions will be available in future releases of OSU INAM. As part of future work, we would like to include application level metrics, enhance the proposed design to collect data from emerging AMD Radeon GPUs, and work with the CUPTI community to resolve the issues with respect to PCI-level data collection.

XI. Acknowledgements

This research is supported in part by NSF grants #CNS-1513120, #ACI-1450440, #CCF-1565414, and #ACI 1664137.

References

[1] Oliver Peckham, "IBMs New Global Weather Forecasting System Runs on GPUs," 2019, Accessed: October 15, 2019. [Online]. Available: https://www.hpcwire.com/2019/01/09/ibm-global-weather-forecasting-system-gpus/

[2] E. Combrisson, R. Vallat, C. O'Reilly, M. Jas, A. Pascarella, A.-l. Saive, T. Thiery, D. Meunier, D. Altukhov, T. Lajnef *et al.*, "Visbrain: A multi-purpose gpu-accelerated open-source suite for multimodal brain data visualization," *Frontiers in Neuroinformatics*, vol. 13, p. 14, 2019.

[3] W. Liu, B. Schmidt, G. Voss, and W. Müller-Wittig, "Molecular dynamics simulations on commodity gpus with cuda," in *High Performance Computing – HiPC 2007*, S. Aluru, M. Parashar, R. Badrinath, and V. K. Prasanna, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 185–196.

[4] NVIDIA, "DGX-2," https://www.nvidia.com/en-us/data-center/dgx-2/, 2016, Accessed: October 15, 2019.

[5] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[6] OSU Micro-benchmarks, http://mvapich.cse.ohio-state.edu/benchmarks/.

[7] A. D. Malony and S. Shende, "Performance Technology for Complex Parallel and Distributed Systems," in *Proc. DAPSYS 2000, G. Kotsis and P. Kacsuk (Eds)*, 2000, pp. 37–46.

[8] HPCToolkit, 2019, Accessed: October 15, 2019. [Online]. Available: http://hpctoolkit.org/

[9] Intel Corporation, "Intel VTune Amplifier," https://software.intel.com/en-us/intel-vtune-amplifier-xe.

[10] "Integrated Performance Monitoring (IPM)," http://ipm-hpc.sourceforge.net/.

[11] "mpiP: Lightweight, Scalable MPI Profiling," http://www.llnl.gov/CASC/mpip/.

[12] "Nagios," http://www.nagios.org/.

[13] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation And Experience," *Parallel Computing*, vol. 30, p. 2004, 2003.

[14] Mellanox Integrated Switch Management Solution, http://www.mellanox.com/page/ib_fabricit_efm_management.

[15] Lawrence Livermore National Laboratory, "PAVE: Performance Analysis and Visualization at Exascale," https://computation.llnl.gov/project/performance-analysis-through-visualization/software.php.

[16] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker, "The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications," ser. SC '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 154–165. [Online]. Available: http://dx.doi.org/10.1109/SC.2014.18

[17] MPI Working Group, "Message Passing Interface Forum," http://www.mpi-forum.org/.

[18] Martin Schulz, "MPIT: A New Interface for Performance Tools in MPI 3," `http://cscads.rice.edu/workshops/summer-2010/slides/performance-tools/2010-08-cscads-mpit.pdf`.

[19] E. Gallardo, J. Vienne, L. Fialho, P. Teller and J. Browne, "MPI Advisor: A Minimal Overhead MPI Performance Tuning Tool," in *EuroMPI 2015*, 2015.

[20] NVIDIA, "CUDA Profiling Tools Interface (CUPTI) ," 2019, Accessed: October 15, 2019. [Online]. Available: https://docs.nvidia.com/cupti/Cupti/r_overview.html

[21] H. Subramoni, A. M. Augustine, M. Arnold, J. Perkins, X. Lu, K. Hamidouche, and D. K. Panda, "INAM 2: InfiniBand Network Analysis and Monitoring with MPI," in *International Conference on High Performance Computing*. Springer, 2016, pp. 300–320.

[22] "MVAPICH: MPI over InfiniBand, Omni-Path, Ethernet/iWARP, and RoCE," Accessed: October 15, 2019. [Online]. Available: http://mvapich.cse.ohio-state.edu/features/

[23] NVIDIA, "Whitepaper: NVIDIA Tesla P100, section 'NVLink High Speed Interconnect'," 2019, Accessed: October 15, 2019. [Online]. Available: https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

[24] B. Barth, T. Evans and J. McCalpin, "Tacc stats," https://www.tacc.utexas.edu/research-development/tacc-projects/tacc-stats.

[25] Virtual Institute - High Productivity Supercomputing, "HOPSA: A Holistic Performance System Analysis," http://www.vi-hps.org/projects/hopsa/overview.

[26] R. Keller, G. Bosilca, G. Fagg, M. Resch, and J. J. Dongarra, "Implementation and Usage of the PERUSE-Interface in Open MPI," in *Proceedings, 13th European PVM/MPI Users' Group Meeting*, ser. Lecture Notes in Computer Science. Bonn, Germany: Springer-Verlag, September 2006.

[27] Barcelona Supercomputing Center, "Paraver," http://www.bsc.es/computer-sciences/performance-tools/paraver.

[28] M. Stephenson, S. K. Sastry Hari, Y. Lee, E. Ebrahimi, D. R. Johnson, D. Nellans, M. O'Connor, and S. W. Keckler, "Flexible software profiling of gpu architectures," *SIGARCH Comput. Archit. News*, vol. 43, no. 3, pp. 185–197, Jun. 2015. [Online]. Available: http://doi.acm.org/10.1145/2872887.2750375

[29] D. Shen, S. L. Song, A. Li, and X. Liu, "Cudaadvisor: Llvm-based runtime profiling for modern gpus," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, ser. CGO 2018. New York, NY, USA: ACM, 2018, pp. 214–227. [Online]. Available: http://doi.acm.org/10.1145/3168831

[30] "OSU INAM," 2019, Accessed: October 15, 2019. [Online]. Available: http://mvapich.cse.ohio-state.edu/tools/osu-inam/